Error Correction Mechanism for Five-Key Chording Keyboards

Adrian Tarniceriu, Bixio Rimoldi, Pierre Dillenbourg
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
adrian.tarniceriu@epfl.ch, bixio.rimoldi@epfl.ch, pierre.dillenbourg@epfl.ch

***Abstract*— As different text input devices lead to different typing error patterns, considering the device characteristics when designing an error correction mechanism can lead to significantly improved results. In this paper, we propose and evaluate a spelling algorithm specifically designed for a five-key chording keyboard. It is based on the maximum a posteriori probability criterion, taking into account a dictionary model and the probabilities that one character is typed for another. These probabilities are determined experimentally. In our experiment, the proposed method reduced the substitution error rate from 7.60% to 1.59%. As comparison, MsWord and iSpell reduced the substitution error rates to 3.12% and 3.94%, respectively.**

Keywords—error correction; confusion matrix; maximum a posteriori probability; chording keyboard

I. INTRODUCTION

Personal computers and mobile computing devices have a constant presence in most people's life, but there are situations when we cannot easily access their services. For example, when walking in a crowded place, the vision should be focused on what happens around us, rather than on typing. Not doing so can have potentially dangerous consequences. Furthermore, input devices such as classic desktop keyboards, keypads or touchscreens may not be suitable for persons that have certain physical disabilities such as limited vision and/or can use only one hand.

Chording keyboards [1] represent a possible solution for the above-mentioned situations. These keyboards allow users to generate a character by simultaneously pressing a combination of keys, similarly to playing a note on a musical instrument. With five keys, there are 31 combinations in which at least one key is pressed, enough for the 26 letters of the English alphabet and five other characters. An additional sixth key can be used to toggle between modes that permit typing capital letters, numbers, or other symbols. If the keys are placed in a position that is naturally under the fingertips (for example on the handlebar of a bike or around a mobile phone), then we can type with only one hand and without looking at the input device. Therefore, we will be able to use a mobile device even during activities for which the visual attention is partially or entirely committed, like walking in crowded spaces, jogging, or riding a bike. Being focused at another activity while typing will probably lead to more errors, so efficient error correction becomes an important issue in these situations.

The most plausible explanation why chording keyboards are not popular is that users require some training before being able to type, to learn the correspondence between key combinations and characters. A previous study [2] showed that people can learn to type with a five-key chording keyboard in less than 45 minutes. According to that study, the average typing rate after 250 minutes of practice is 15.2 words per minute (wpm) with a maximum of 19.2 wpm, comparable to iPhone or Twiddler [3] rates. The error rate is low, 0.22%, because users preferred to correct their mistakes. An effective mean of automatically correcting these mistakes might increase the keyboard's ease-of-use and typing speed because users will not have to stop typing in order to correct errors.

This paper represents an initial effort to explore the area of error correction for chording keyboards, by taking into consideration the particularities of the text input device. This is motivated by the fact that different devices lead to different error patterns, and knowledge about these patterns can be used to improve the error correction methods. The error correction method that we propose is based on the maximum a posteriori probability principle (MAP) [4]. For every typed word, it provides a list of possible candidates and chooses the one that is the most likely. The correction method is developed for a five-key keyboard, but it can be easily generalized to other designs.

The paper is organized as follows. In Section II, we overview existing text error correction mechanisms. In Section III, we describe the proposed error correction algorithm. In Section IV, we describe the data set used to evaluate the algorithm, and in Section V, we present the results. In Section VI, we conclude the paper and discuss future directions.

II. RELATED WORK

Traditionally, text error detection and correction focuses on character-level errors, which can be classified into three categories: deletions, when a character is omitted, insertions, when an additional character is inserted, and substitutions, when a character is substituted by another character.

Some approaches take into account the context, grammatical and semantical rules, and also detect errors such as missing words, wrong phrase structure, misused inflections, or others.

A detailed overview of commonly used correction techniques is presented by Kukich in [5]. Research in spelling error detection and correction is grouped in three main categories:

1. Non-word error detection:

Groups of *n* letters (*n*-grams) are examined and looked up in a table of statistics. The strings that contain non-existing or highly infrequent *n* -grams are considered errors.

2. Isolated word error correction:

Each word is treated individually and considered either correct or incorrect. In the latter case, the incorrectly spelled word is compared to entries from a dictionary. Based on similarities between the typed word and dictionary words, a list of possible candidates is proposed. These candidates can be provided using several techniques:

- minimum edit distance techniques consider the minimum number of editing operations required to transform a string into another. A basic example is to consider the dictionary word that can be obtained from the typed word with a minimum number of insertions, deletions and substitutions;
- similarity key techniques map each string to a key which is similar or identical for similarly spelled strings. In this way, the key for a misspelled string can point to similarly spelled candidates from the dictionary. The advantage of this approach is that the misspelled string is not compared to all entries in the dictionary;
- rule-based techniques propose candidate words by using knowledge of the most common errors;
- probabilistic techniques, which consider transition and confusion probabilities. The first ones provide the probability that a letter is followed by another given letter (the values are language dependent). Confusion probabilities estimate how often a letter is typed instead of another letter (the values are text-input device dependent);
- among other possible methods, *n*-gram techniques and neural net techniques can also be efficiently used.

Most isolated word error correction methods do not correct errors when the erroneously typed word is contained in the dictionary. For example, if *farm* was typed instead of *form*, no error will be detected. Moreover, these methods cannot detect the use of wrongly inflected words (for example, *they is* instead of *they are*).

3. Context dependent error correction:

These methods try to overcome the drawbacks of analyzing each word individually by also considering the context. Errors can be detected by parsing the text and identifying incorrect part-of-speech or part-of-sentence *n*-grams. Or, if enough memory and processing power are available, tables of word *n*-grams can be used. Other approaches consider grammatical and inflectional rules, semantical context, and can also identify stylistic errors.

Most of the methods presented above can be applied to any typed text, regardless of the input device. As various input techniques become more and more popular, the classic correction techniques have been improved to consider both the text and the device particularities. Goodman et al. [6] presented an algorithm for soft keyboards that combines a language model and the probabilities that the user hits a key outside the boundaries of the desired key. Kristensson and Zhai [7] proposed an error correction technique for stylus typing using geometric pattern matching. The T9 text input method for mobile phones can also be included here, as it considers the correspondence between keys and characters to predict words.

Sandnes and Huang classify chording errors in three categories: deletions, when the user does not press one of the required keys, insertions, when the user presses an extra key, and substitutions, when the user makes a mistake between adjacent fingers. Assuming that most words have very few errors, they propose an algorithm for chording text input that can correct words that contain one deletion, insertion, or substitution [8].

III. Error Correction Algorithm

The proposed error correction method corrects character substitution errors and focuses on individual words, without considering any contextual information. It is designed for a chording keyboard with five keys, each key being operated by a finger of the right hand. The algorithm is based on the maximum a posteriori probability (MAP) principle, and for every typed string, it finds the string that is the most likely to be typed and is a valid word.

We can interpret the typing process as sending information over a communication channel. The symbol at the channel input, x, is the word to be typed and the channel output, y, is what has actually been typed. The MAP algorithm will find the string $\hat{x}$, which is the most likely in the sense of maximizing the posterior probability $p(x \mid y)$ over all $x \in S$. The set S contains all the possible candidate strings. If we denote by $p(x)$ and $p(y)$ the distributions for the channel input and output respectively, then

$$\begin{aligned} \hat{x} &= \arg\max_{x \in S} p(x \mid y) \\ &= \arg\max_{x \in S} \frac{p(y \mid x)p(x)}{p(y)} \\ &= \arg\max_{x \in S} p(y \mid x)p(x). \end{aligned} \tag{1}$$

Because our goal is to design a spelling algorithm, we can reduce the set of candidates from all possible strings to dictionary words. Moreover, as we focus on substitutions, we can limit the candidate set to words with the same length as the typed word. Considering this and assuming that the typing of each letter is an independent event, we can write

Fig. 1. Examples of letter mappings: "**i**" is given by the initial of the finger pressing the key (index). "**m**" is given by the shape of the fingers pressing the keys. "**w**" is given by the shape of the fingers not pressing the keys. For "**x**", we represent the four corners, and for "**o**", we imagine five dots spread around a circle.

$$p(y \mid x) = \prod_{i=1}^{N} p(y_i \mid x_i), \tag{2}$$

where y_i is the *i*-th letter of the typed word, x_i is the intended letter, and N is the word length. The conditional probability $p(y_i \mid x_i)$ is the probability that the character y_i is typed in lieu of x_i. The prior probability, $p(x)$, is given by the frequencies of the dictionary entries in the English language. For example, given the typed word $y = oat$ and the candidate $x = bat$, we need to compute

$$p(oat \mid bat)p(bat) = p(o \mid b)p(a \mid a)p(t \mid t)p(bat). \tag{3}$$

Determining the posterior probabilities for all dictionary words with the same length as the typed word can be too computationally demanding. Therefore, to reduce the complexity, we use the fact that only a certain fraction of the substitutions occur with non-negligible probability. To describe how this is done, it is useful to represent each character by a five-bit codeword. We choose the first digit to represent the key under the thumb, the second to represent the key under the index, etc. The value of a position is 1 if the corresponding key is pressed and 0 otherwise. So, for instance, the sequence 11011, corresponding to the letter "**x**", means that all fingers except the middle are pressing the keys. In this way, we can compare two words not only by the edit distance, but also from a bit distance point of view.

The key-to-character mapping was designed to minimize the learning time by assigning intuitive mnemonics to each character. In Fig. 1, we show five examples, and the complete list is given in the Appendix.

Our tests have shown that in 98.5% of the cases, the wrongly typed word differs from the intended one by at most five bits. Hence, for each typed word we limit the set of possible candidates, S, to words with the same length and which differ by at most five bits. Four of the possible candidates for the typed word *oat* are given in Table I. *ham* is not a valid candidate, because it differs by six bits.

In our study, we use the British National Corpus, containing approximately 100 million words [9]. The used dictionary was obtained from this corpus by choosing all the items occurring at least five times. It contains 100,944 entries, which include inflected forms such as declensions and conjugations. The probabilities $p(x)$ are given by the word frequency in the corpus and the confusion probabilities, $p(y_i \mid x_i)$, were estimated experimentally.

TABLE I. POSSIBLE CANDIDATES FOR THE TYPED WORD *OAT*

Possible candidates	Binary form	Bit distance
oat	11111 00110 10000	0
bat	10111 00110 10000	1
rat	00010 00110 10000	4
ore	11111 00010 11000	2
ham	11001 00110 01110	6-not valid

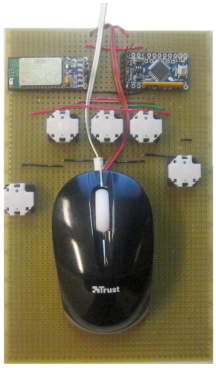

Fig. 2. Chording keyboard prototype

IV. EXPERIMENTAL SETUP AND EVALUATION DATA

In order to gather enough data to evaluate the proposed algorithm, we asked ten PhD students from our university (eight male and two female, with ages between 24 and 31) to type using a chording keyboard. All are right handed and have previously participated in another typing study [2], so they already knew the mapping. They used a five-key chording keyboard prototype with the keys placed around a computer mouse, presented in Fig. 2.

We designed the prototype in this way because we wanted the subjects to see a practical application of a chording device, allowing typing and screen navigation at the same time, with only one hand. The keyboard is designed using an Arduino Pro Mini microcontroller board and communicates with the computer by Bluetooth. The buttons are placed so that they can be easily operated while holding the mouse with the palm. We used keys and not pressure or touch sensors because they provide a distinct tactile feedback.

The participants were asked to type for 10 sessions of 30 minutes each, while sitting at a desk. Each session consisted of three rounds of 10 minutes, separated by breaks of two minutes. In the beginning of each round, the participants warmed up by typing each letter of the alphabet. During the warm-up, a help image showing the key combination for the letter to be typed was displayed. Afterwards, the help image was no longer available and the participants typed sentences chosen from a set considered representative for the English language [10]. These sentences were pre-prepared before the experiment to contain only small letters and no punctuation signs.

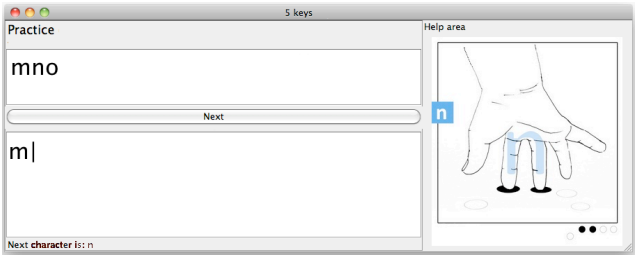

Fig. 3. Screenshot of the application used for typing

A Java application was designed to display the text to be typed and to monitor the pressed keys. A screenshot of the application is shown in Fig. 3. The top-left window contains the text to be typed and the bottom-left window represents the typing area. The help image is displayed on the right.

Because we wanted to evaluate an error correction mechanism, we instructed the participants not to correct their mistakes (however, this was not enforced and they could delete typed text). As a reward for the time commitment during the experiment, they received a fixed monetary compensation for the first nine typing sessions. To provide additional motivation, for the last session, the reward was proportional to the number of typed words and to the typing accuracy.

In general, typing errors can be classified into cognitive and sensorimotor errors. Cognitive errors appear when users type a word wrongly because they do not know the correct spelling or they do not remember the key combination for a certain character, while sensorimotor errors appear when a coordination mistake is produced during the execution. During the experiment, the text to be typed was shown to the users, and they already knew the mapping, so we can assume that most errors are sensorimotor. Therefore, the errors will be more dependent on the input device and mapping than on the user's knowledge of the mapping.

The total amount of data gathered during the experiment consists of 40,345 words. Out of these, 4052 (10.17%) contain errors. 3065 (75.64%) of the errors are substitution errors when one or more letters of the word are replaced by other letters (for example *hous**s*** instead of *hous**e***). The remaining 987 errors occurred when people did not type a letter (*hous* instead of *hous**e***), typed an extra letter (*hous**ee*** instead of *hous**e***), the space between two consecutive words was missing (*thehouse* instead of *the house*), they combined several of the above-mentioned mistakes, or when whole words were missing, added, or the topic of the sentence changed.

The total number of typed characters is 219,308. We used these characters to determine the confusion matrix, which is a square matrix with rows and columns labeled with all the characters that can be typed. The value at position *ij* shows the frequency of character *j* being typed when *i* was intended. The values are given as percentages from the total number of occurrences for character *i* and will represent the conditional probabilities, $p(j\,|\,i)$. In Table II, we present the values of the confusion matrix for the characters from *a* to *e*.

TABLE II. CONFUSION MATRIX ENTRIES FOR A, B, C, D, AND E

%	*a*	*b*	*c*	*d*	*e*
a	96.79	0.01	0.03	0.02	0.86
b	0.01	94.17	0.01	1.17	0.07
c	0.01	0.01	98.14	0.01	0.90
d	0.01	0.47	0.03	95.37	0.03
e	0.11	0.01	0.17	0.03	97.92

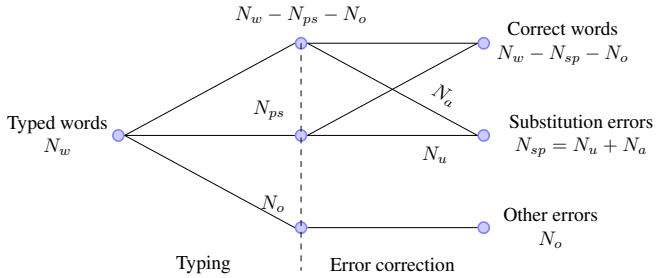

Fig. 4. Typing and error correction diagram

By analyzing the confusion matrix and the five-bit code for each letter, we notice that 44.81% of the wrongly typed characters differ from the intended character by one bit, 39.22% by two bits, 8.72% by three bits, 5.87% by four bits and 1.39% by five bits. If we check word by word and consider only substitution errors, 91.48% of the erroneous words contain one substitution, 7.68% contain two substitutions and 0.67% three substitutions. From a bit-error point of view, 40.56% of the erroneous words contain a one-bit error, 40.92% a two-bit error, 7.62% a three-bit error, 7.20% a four-bit error and 2.13% a five-bit error.

V. RESULTS

The typing and error correction processes are described by the diagram of Fig. 4. The three vertical levels represent correct words, substitution errors and other error types, respectively. In the following, we will refer to the error rates before applying the proposed algorithm as pre-processing error rates, and to the error rates obtained after applying the algorithm as post-processing error rates.

The left part of the diagram represents the typing process. Out of N_w (40,365) typed words, N_{ps} (3065) are substitution errors, N_o (987) represent other error types, and the rest are correct. The "*p*" before "*s*" in the subscript of N_{ps} stands for pre-processing. The pre-processing substitution error rate is computed as N_{ps} / N_w, and the pre-processing overall error rate as $(N_{ps} + N_o) / N_w$.

The right part of the diagram shows the error correction process. The final number of correct words is made up of corrected substitution errors and words that were correctly spelled from the beginning. The number of substitution errors remaining after processing is denoted by N_{sp}, where the "*p*"

TABLE III. SUBSTITUTION AND OVERALL ERROR RATES BEFORE AND AFTER APPLYING THE ALGORITHM

	Pre-processing	**Post-processing**				
		MAP algorithm	***MsWord***	***iSpell***	***MsWord MAP***	***iSpell MAP***
Substitution error rates %	7.60	1.59	3.12	3.94	1.77	1.58
Overall error rates %	10.04	4.04	5.57	6.38	4.22	4.03

after "*s*" stands for post-processing. N_{sp} has two components: N_u, the substitution errors uncorrected by the algorithm, and N_a, additionally introduced substitution errors (any algorithm that determines the most likely candidate will occasionally introduce new errors). The number of other error types, N_o, is not affected by the algorithm. The final number of errors is $N_e = N_{sp} + N_o$. The post-processing substitution error rate is computed as N_{sp} / N_w, and the post-processing overall error rate as $(N_{sp} + N_o) / N_w$.

The error-correction algorithm was implemented in MATLAB. For evaluation, we compared the pre-processing and post-processing error rates. We also compared the proposed algorithm to MsWord and iSpell. MsWord and iSpell return an ordered list of candidates from which the user should choose, the first one being the most likely. These algorithms can also return candidate words with different lengths. To be consistent in comparing the three algorithms, we only considered the candidates with the same length as the typed word. This is fair because we are only analyzing substitution errors. The first one of these candidates is taken as the chosen word.

In addition to determining the various error rates for the processed text, we also compared the candidate sets provided by the proposed algorithm, MsWord and iSpell. The first one is based on bit distance, while the last two on edit distance. These sets were compared by applying the MAP algorithm to the candidates provided by MsWord and iSpell. The results are presented in Table III.

The proposed error correction method reduces the substitution error rate from 7.60% to 1.59%, which is considerably better than the rates for MsWord and iSpell (3.12% and 3.94%, respectively). The results are also better than applying the MAP algorithm to the MsWord candidates and very close to applying it to the iSpell candidates, proving the bit distance to be an efficient metric when building the candidate set. However, one should not forget that the dictionaries are not the same, and this can affect the results. Moreover, our algorithm is specifically designed for a five key chording keyboard, while MsWord and iSpell can be applied to any text input device with the same results.

The proposed algorithm corrects significantly more substitution errors than MsWord and iSpell, but it is unavoidable that some correctly typed words are modified. This occurs if another word is more likely given the typed word. For example, if we type *tee*, which is a valid English word contained in the dictionary, it will be changed to *the*. This can be an important inconvenient, because, in general, users are less displeased when the algorithm does not correct an error than when the algorithm modifies a correctly typed word.

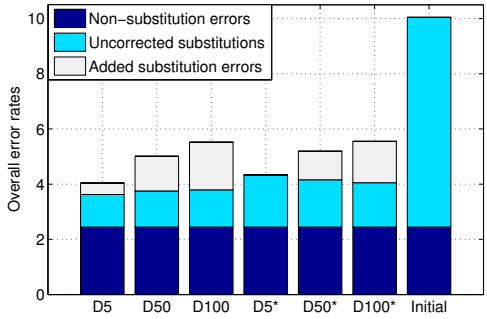

Fig. 5. Overall error rates for different dictionaries

One may be tempted to reduce the number of additional errors by accepting as correct the words that are found in the dictionary. This method is no longer a MAP algorithm, therefore we expect the overall error rates to increase. We tested this by using three dictionaries denoted by D5, D50 and D100. The number next to the letter D is the minimum number of appearances of every dictionary entry in the used corpus. All of them contain inflected words and their sizes are 100,944, 41,028, and 11,288 words, respectively.

For each of these dictionaries, we computed the number of uncorrected substitution errors and the number of newly introduced errors (N_u and N_a) for two cases: with the original and with the modified algorithm. A graphical representation of the contribution of each error type to the overall error rates is given in In Fig. 5, and the detailed results are given in Table IV. A * sign next to the dictionary name means that the dictionary words were not modified. The results confirm our expectations: modifying the algorithm reduces the additional errors, but also corrects fewer of the existing ones, leading to higher overall error rates.

We can also notice that smaller dictionaries lead to higher error rates. From a fundamental point of view, reducing the dictionary size by excluding words with low frequency in the corpus is equivalent to setting their prior probabilities to zero. Using less accurate priors can only reduce the performance of the algorithm, thus increasing the error probability.

TABLE IV. UNCORRECTED, ADDED AND REMAINING SUBSTITUTION ERRORS

	Pre-processing		Post-processing				
	Substitution errors	***Other errors***	***Uncorrected substitutions***	***Added substitutions***	***Substitution errors***	***Substitution error rate %***	***Overall error rate %***
D5	3065	987	476	167	643	1.59	4.04
D50			527	508	1035	2.56	5.01
D100			544	696	1240	3.07	5.52
D5*			762	1	763	1.89	4.34
D50*			691	417	1108	2.74	5.19
D100*			648	605	1253	3.10	5.55

VI. CONCLUSION

In this paper, we have presented an error correction algorithm for substitution errors, designed for a five-key chording keyboard. For every typed word, it selects several possible candidates and returns the most likely one using the MAP algorithm. This method decreases the substitution error rate from 7.60% to 1.59%, providing a considerable improvement compared to MsWord and iSpell (leading to substitution error rates of 3.12% and 3.94%, respectively). This advantage is due to the MAP algorithm, which takes into account the prior distribution of words and the confusion probabilities, which depend on the input device. Even if it was designed for a specific keyboard and mapping, the presented method can be easily generalized to other input devices by updating the confusion matrix.

We have only focused on substitution errors because they represent more than 75% of the total errors. Given the encouraging results, as a next step we will expand the algorithm to also consider other error types such as missing or extra characters. The performance of the MAP algorithm can be improved by increasing the accuracy of the confusion matrix and of the prior probabilities. One possibility is to implement an adaptive approach, starting with a common matrix and updating it for individual users based on what they type. Words that are typed more often can have their prior probability increased. One should also be able to add new words to the dictionary.

Even though it is not the purpose of this study, the gathered data enables us to estimate the achievable typing rates. Taking into account the previous experience of the participants, after approximately 350 minutes of typing, the average speed is 20.1 wpm, with the maximum of 31.2 wpm. The average character error rate is 2.91%, but the users were instructed not to correct errors. Considering this, the chording keyboards can be a viable typing option, especially in situations where one wants to type a short text and cannot continuously look at the keys, such as walking in crowded places or riding a bike. In addition, they can facilitate text input for persons who can only use one hand – the prototype used in this study allows a person to type and control the mouse with one hand and without the need for moving the hand from the keyboard to the mouse and back. Because one does not need to move the fingers from one key to another, chording keyboards can also be used by visually impaired persons. Efficient error correction will only increase the usability of these keyboards.

APPENDIX

TABLE V. KEY COMBINATIONS FOR THE USED CHARACTERS

Character	**5-bit code**	**Character**	**5-bit code**
a	00110	*q*	01101
b	10111	*r*	00010
c	10100	*s*	10101
d	11101	*t*	10000
e	11000	*u*	01001
f	01010	*v*	10011
g	11100	*w*	10001
h	11001	*x*	11011
i	01000	*y*	10110
j	01011	*z*	10010
k	11010	*space*	11110
l	00111	*backspace*	01111
m	01110	*enter*	00011
n	01100	*period*	00100
o	11111	*comma*	00101
p	00001		

References

[1] J. Noyes, "Chord keyboards," *Applied Ergonomics,* vol. 14, no. 1, 1983, pp. 55 – 59.

[2] A. Tarniceriu, P. Dillenbourg, and B. Rimoldi, "Single-handed typing with minimal eye commitment: A text-entry study," in *The Sixth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies,* September 2012, pp. 117 – 122.

[3] K. Lyons, et al., "Twiddler typing: one-handed chording text entry for mobile phones," in Proceedings of the SIGCHI conference on Human factors in computing systems, CHI '04, (Vienna, Austria), ACM, 2004, pp. 671–678.

[4] S. Kay, "Fundamentals of statistical signal processing: estimation theory," Prentice-Hall, 1993.

[5] K. Kukich, "Techniques for automatically correcting words in text," *ACM Comput. Surv. 24,* 4 , December 1992, pp. 377–439.

[6] J. Goodman, G. Venolia, K. Steury, and Parker, "C Language modeling for soft keyboards, " in *Proceedings of the 7th international conference on Intelligent user interfaces,* IUI '02, (New York, NY, USA), ACM, 2002, pp. 194–195.

[7] P. O. Kristensson, and P. Zhai, "Relaxing stylus typing precision by geometric pattern matching," in *Proceedings of the 10th international conference on Intelligent user interfaces,* IUI '05, (New York, NY, USA), ACM, 2005, pp. 151–158.

[8] F. Sandnes, and Y. P. Huang, "Non-intrusive error-correction of text input chords: a language model approach," in *Fuzzy Information Processing Society, 2005. NAFIPS 2005,* June 2005, pp. 373 – 378.

[9] http://www.kilgarriff.co.uk/bnc-readme.html, July 2013.

[10] I. S. Mackenzie and R. W. Soukoreff, "Phrase sets for evaluating text entry techniques," in *Extended Abstracts of the ACM Conference on Human Factors in Computing Systems CHI '03,* (Fort Lauderdale, Florida, United States), ACM, 2003, pp. 766– 767.